VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Object Oriented Java Programming

(23CS3PCOOJ)

Submitted by

ABHINAV C (1BM23CS008)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019

Sep-2024 to Jan-2025

B.M.S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Object Oriented Java Programming (23CS3PCOOJ)" carried out by **StudentName (1BM23CS000),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Object Oriented Java Programming (23CS3PCOOJ) work prescribed for the said degree.

Lab faculty Incharge Name Assistant Professor Department of CSE, BMSCE	Dr. Jyothi S Nayak Professor & HOD Department of CSE, BMSCE

Index

<u>Github Lin</u>k:
https://github.com/Abhinav-2013

<u>Program 1</u>
Implement Quadratic Equation

<u>Algorithm:</u>
<u>1. Initialization</u>
Start the program.
Import the java.util.Scanner package for user input
<u>2.Attributes:</u>
Define integer variables a, b, and c to store the coefficients of the quadratic equation.
Define double variables r1, r2 to store the roots, and d_sq to store the square root of the discriminant.
Create a Scanner object sc to handle user input
<u>3.Methods:</u>
input():
Prompt the user to enter coefficients aa, bb, and cc.
Read the values using sc.nextInt() and store them in a, b, and c.
calc():
Compute the discriminant:D=b2−4acD=b2−4ac
Print the value of DD for verification.
Based on the value of DD:
If D=0D=0:
Compute the roots as:r1=r2=−b2ar1=r2=2a−b
Print "Roots are real and equal."
Display the roots r1r1 and r2r2.
If D>0D>0:
Compute the square root of DD:d_sq=Dd_sq=D
Compute the roots as:r1=−b+d_sq2a,r2=−b−d_sq2ar1=2a−b+d_sq,r2=2a−b−d_sq
Print "Roots are real and distinct."
Display the roots r1r1 and r2r2.
If D<0D<0:
Compute the square root of the absolute value of DD:d_sq=−Dd_sq=−D
Compute the real and imaginary parts of the roots as:r_real=−b2a,r_imaginary=d_sq2ar_real=2a−b,r_imaginary=2ad_sq
Print "Roots are imaginary."
Display the roots in the

form:r1=r_real+r_imaginaryi,r2=r_real−r_imaginaryir1=r_real+r_imaginaryi,r2=r_real−r_imaginary i

4.Define the Quadratic Class

main() Method:
Print the programmer's name and USN.
Create an object quad of the Quad class.
Call the input() method of the quad object to take input for coefficients.
Call the calc() method of the quad object to compute and display the roots.

5. End Program

Exit the program

Code:

```
import java.util.*;
class Quad{
Scanner sc=new Scanner(System.in);
int a,b,c,d;
double r1,r2,d_sq;
void input(){
System.out.println("Enter coefficients a,b,c:");
a=sc.nextInt();
b=sc.nextInt();
c=sc.nextInt();
}
void calc(){
int d=b*b-4*a*c;
System.out.println(d);
if (d==0){
r1=-b/(2.0*a);
System.out.println("Roots are real and equal");
System.out.println("Root 1 = "+r1+"\nRoot 2 = "+r1);
}
else if(d>0){
r1=(-b+d_sq)/(2.0*a);
r2=(-b-d_sq)/(2.0*a);
System.out.println("Roots are real and distinct");
System.out.println("Root 1 = "+r1+"\nRoot 2 = "+r2);
}
else{
```

```
d_sq=Math.sqrt(-d);
r1=-b/(2.0*a);
r2=d_sq/(2.0*a);
System.out.println("Roots are imaginary");
System.out.println("Root 1 = "+r1+" + "+r2+"i"+"\nRoot 2 = "+r1+" - "+r2+"i");
}
}
}
class Quadratic{
public static void main(String[] args){
System.out.println("Name: Abhinav C \nUSN: 1BM23CS008");
Quad quad=new Quad();
quad.input();
quad.calc();
}
}
```

Program 2
SGPA calculator
Algorithm:
1. Initialization
Start the program.
Import the necessary library (java.util.Scanner) for user input
2. Define the Student Class

Declare attributes:

String name to store the student's name.
String usn to store the student's university roll number.
Arrays int credits[8] and int marks[8] to store subject credits and marks respectively.
double sgpa to store the SGPA for a semester.
double cgpa to store the CGPA after two semesters.
int grade[8] to store grades for the subjects
Define Methods:

input():
Use a Scanner object to get input for the credits and marks for 8 subjects.
calculate(int marks[], int credits[]):
Initialize variables sum (to store total weighted grades) and div (to store total credits).
For each subject:
Calculate the grade using the formula:Grade={Marks+1010if Marks ≠10010if Marks =100Grade={10Marks+1010if Marks =100if Marks =100
Add the weighted grade to sum.
Add the credits to div.
Compute SGPA using the formula:SGPA=sum of weighted gradessum of creditsSGPA=sum of creditssum of weighted grades
calcgpa(double sgpa1, double sgpa2):
Compute CGPA as the average of two semester SGPAs:CGPA=SGPA1+SGPA22CGPA=2SGPA1+SGPA2

3. Main Program Execution

Create a Scanner object for input.

Input Number of Students:

Prompt the user to enter the number of students, n.
Create an array of Student objects with size n.

Iterate for Each Student:

For each student, perform the following:
Input Student Details:
Prompt the user to enter the student's name and USN.
Store these details in the corresponding Student object.
Input Semester 1 Details:
Call the input() method to collect subject credits and marks.
Call calculate() to compute SGPA for Semester 1.
Input Semester 2 Details:
Repeat the above steps for Semester 2.
Calculate CGPA:
Call calcgpa() with the SGPAs of both semesters to compute CGPA.
Display Results:
Print the SGPA for both semesters and the CGPA for the student.

4. End Program
After processing all students, exit the program.

Code:
```
import java.util.Scanner;

class Student
{
   String name;
   String usn;
   int credits[] = new int[8];
   int marks[] = new int[8];
   double sgpa=0.0;
   double cgpa;
   int grade[] = new int[8];

   double calculate(int m[], int c[])
  {
      int j;
      double sum = 0.0;
      int div = 0;
      for (j = 0; j < 8; j++)
      {
         if (m[j] != 100)
         {
            grade[j] = (m[j] + 10) / 10;
         }
         else
         {
            grade[j] = 10;
          }
          div = credits[j] + div;
          sum = sum + (grade[j] * credits[j]);
          System.out.println("Grade for subject " + (j + 1) + ": " + grade[j]); // error check
        }
        sgpa = sum / div;
        System.out.println("SGPA: " + sgpa);
        return sgpa;
    }
double calcgpa(double sgpa1, double sgpa2)
{
cgpa=(sgpa1+sgpa2)/2;
return cgpa;
}
void input()
{
Scanner sc=new Scanner(System.in);
        System.out.println("Now enter subject credits for semester:");
        int i;
        for (i = 0; i < 8; i++)
        {
          credits[i] = sc.nextInt();
        }
        System.out.println("Now enter subject marks for semester:");
        for (i = 0; i < 8; i++)
        {
          marks[i] = sc.nextInt();
        }
}
public static void main(String args[]) {
Scanner sc1 = new Scanner(System.in);
System.out.println("Enter number of students: ");
int n=sc1.nextInt();
Student obj[]=new Student[n];
int k;
for(k=0;k<n;k++)
{
obj[k]=new Student();
System.out.println("Enter Student name: ");
String name = sc1.next();
System.out.println("Enter Student USN: ");
String usn = sc1.next();
obj[k].input();
```

```
System.out.println("Semester 1");
double result = obj[k].calculate(obj[k].marks, obj[k].credits);
System.out.println("1st Semester SGPA for " + obj[k].name + " (" + obj[k].usn + ") is: " +
result);
System.out.println("Semester 2");
obj[k].input();
double result2 = obj[k].calculate(obj[k].marks, obj[k].credits);
System.out.println("2nt Semester SGPA for " + obj[k].name + " (" + obj[k].usn + ") is: " +
result2);
System.out.println("CGPA for 1st year is : "+obj[k].calcgpa(result,result2));
}
}
}
```

<u>Program 3:</u>
Book details

<u>Algorithm:</u>

<u>1. Initialization</u>
Start the program.
Import the java.util.Scanner class for taking input.

<u>2. Define the Book Class</u>

<u>Attributes:</u>

String name: The name of the book.
String author: The author of the book.
double price: The price of the book.
int numPages: The number of pages in the book.

<u>Constructor:</u>

Define a constructor with the parameters name, author, price, and numPages to initialize the book object.

<u>Setter Methods:</u>

setName(String name): Set the book's name.
setAuthor(String author): Set the book's author.
setPrice(double price): Set the book's price.
setNumPages(int numPages): Set the number of pages.

<u>Getter Methods:</u>

getName(): Return the book's name.
getAuthor(): Return the book's author.
getPrice(): Return the book's price.
getNumPages(): Return the number of pages.

<u>toString() Method:</u>

Override the toString method to return the book's details as a formatted string.

3. Define the BookMain Class

Attributes:

Declare a Scanner object for input.

Main Method:

Print the programmer's name and USN.

Input the Number of Books:

Prompt the user to enter the number of books n.
Use scanner.nextInt() to read the input and store it in n.
Create an array books of type Book with size n.

Input Book Details:

Iterate from 0 to n-1:
Prompt the user to enter details for each book:
Book name (read using scanner.nextLine()).
Author name (read using scanner.nextLine()).
Price (read using scanner.nextDouble()).
Number of pages (read using scanner.nextInt()).
Create a Book object using the constructor and store it in the books array.

Display Book Details:

Print "Book Details:".
Iterate through the books array and print each book's details using the toString() method.

Close Scanner:

Close the Scanner object to release resources.

4. End Program
Exit the program.

Code:
```
import java.util.Scanner;
class Book {
private String name;
```

```
 private String author;
private double price;
 private int numPages;
public Book(String name, String author, double price, int numPages)
{
this.name = name;
this.author = author;
this.price = price;
 this.numPages = numPages;
 }

public void setName(String name) {
 this.name = name;
 }
public void setAuthor(String author)
 {
this.author = author;
 }
 public void setPrice(double price)
 {
this.price = price;
}
public void setNumPages(int numPages)
 {
this.numPages = numPages;
 }
public String getName()
 {
return name;
 }
public String getAuthor()
 {
 return author;
 }
public double getPrice()
{
 return price;
}
 public int getNumPages()
{
return numPages;
 }
// toString method
@Override
```

```
 public String toString()
{
return "Book Name: " + name + ", Author: " + author + ", Price: " + price + ", Number of
Pages: " + numPages;
}
}
public class BookMain { public static void main(String[] args)
{
System.out.println("Name: Abhinav C\nUSN : 1BM23CS008");
 Scanner scanner = new Scanner(System.in);
System.out.println("\nEnter the number of books: ");
int n = scanner.nextInt();
 scanner.nextLine();
Book[] books = new Book[n];
 for (int i = 0; i < n; i++)
{ System.out.println("Enter details for book " + (i + 1) + ":");
 System.out.print("Name: ");
String name = scanner.nextLine(); System.out.print("Author: ");
String author = scanner.nextLine(); System.out.print("Price: ");
double price = scanner.nextDouble();
System.out.print("Number of Pages: ");
 int numPages = scanner.nextInt(); scanner.nextLine();
// Consume the newline
 books[i] = new Book(name, author, price, numPages);
}
 System.out.println("\nBook Details:");
for (Book book : books)
{
System.out.println(book);
}
 scanner.close();
 }
}
```

Program 4
Area of shape

Algorithm
1. Define Abstract Class Shape
Declare an abstract class named Shape.
Attributes:
dimension1 and dimension2 (both int):
Represent dimensions required to compute the area of shapes.
For some shapes (like Circle), dimension2 may not be used.
Abstract Method:
Declare an abstract method printArea(), which is meant to be implemented by subclasses to compute and display the area of the shape.

2. Define Subclasses

Rectangle Class:

Extend the Shape class.
Constructor:
Accept length and width as parameters and initialize dimension1 and dimension2.
Method Implementation:
Implement printArea() to calculate the area of a rectangle:Area=length×widthArea=length×width

Triangle Class:

Extend the Shape class.
Constructor:
Accept base and height as parameters and initialize dimension1 and dimension2.

Method Implementation:
Implement printArea() to calculate the area of a triangle:Area=0.5×base×heightArea=0.5×base×height

Circle Class:

Extend the Shape class.
Constructor:
Accept radius as a parameter and initialize dimension1 (ignoring dimension2).
Method Implementation:
Implement printArea() to calculate the area of a circle:Area=π×radius2Area=π×radius2
Use Math.PI for the value of ππ.

3. Main Class ShapeTest

Create Objects:

Create objects for each subclass (Rectangle, Triangle, and Circle) using their respective constructors.
Use polymorphism by declaring the object references as type Shape.

Call printArea():

Invoke the printArea() method on each object to compute and display the respective areas

program:

```
abstract class Shape {
   protected int dimension1;
   protected int dimension2;


   public abstract void printArea();
}


class Rectangle extends Shape {
   public Rectangle(int length, int width) {
      this.dimension1 = length;
      this.dimension2 = width;
```

```
    }


    public void printArea() {
        int area = dimension1 * dimension2;
        System.out.println("Area of Rectangle: " + area);
    }
}


class Triangle extends Shape {
    public Triangle(int base, int height) {
        this.dimension1 = base;
        this.dimension2 = height;
    }


    public void printArea() {
        double area = 0.5 * dimension1 * dimension2;
        System.out.println("Area of Triangle: " + area);
    }
}


class Circle extends Shape {
    public Circle(int radius) {
        this.dimension1 = radius;
    }

    public void printArea() {
        double area = Math.PI * dimension1 * dimension1;
        System.out.println("Area of Circle: " + area);
    }
}


public class ShapeTest {
    public static void main(String[] args) {

        Shape rectangle = new Rectangle(5, 3);
        rectangle.printArea();

        Shape triangle = new Triangle(4, 6);
        triangle.printArea();
        Shape circle = new Circle(7);
        circle.printArea();
    }
}
```

Program 5:
Bank

Algorithm for the Bank Account Program
This program models a simple banking system where a user can interact with Savings and Current accounts. Below is the step-by-step algorithm:

1. Initialize the Classes
Bank Class (Base Class)

Define attributes:

accountNo: Integer, stores the account number.
balance: Double, stores the account balance (default = 0).

Define methods:

Bank(int accountNo): Constructor to initialize accountNo and set balance to 0.
deposit(double depAmount): Adds depAmount to the balance.
withdraw(double withAmount): Deducts withAmount from the balance.
interest(double rate, int time):
Prints a message saying "Interest is not applicable in current account".
Returns 0.0.

SavingsAccount Class (Inherits from Bank)

Define constructor:

SavingsAccount(int accountNo): Calls the Bank constructor to initialize accountNo.

Override the interest() method:

Calculate compound interest:Interest=balance×(1+rate100)time−balanceInterest=balance×(1+100rate)time−balance
Add the calculated interest to the balance.
Return the interest.

CurrentAccount Class (Inherits from Bank)

Define static attribute:

withdrawLimit: Double, sets a withdrawal limit (default = 1000).

Define constructor:

CurrentAccount(int accountNo): Calls the Bank constructor to initialize accountNo.

Override the withdraw() method:

Deduct withAmount from the balance.
If the balance falls below the withdrawLimit:
Print a warning: "Withdraw Limit Reached - Deducting Service Charge".
Deduct a service charge of 100 from the balance.

2. Main Class: Run
Step 1: Print Header
Display:

Copy code

Abhinav C1BM23CS008

Step 2: Account Type Selection
Prompt the user to choose an account type:

sql

Copy code

1. Open Savings Account2. Open Current Account

Take the user's input:
If the choice is 1, create a SavingsAccount object.
If the choice is 2, create a CurrentAccount object.
Step 3: Display Menu Options
Show the following menu:

markdown

Copy code

1. Deposit2. Withdraw3. Show Balance4. Compute Interest5. Exit

Step 4: Perform Operations in a Loop
Prompt the user for a menu choice.
Perform actions based on the user's choice:
Case 1: Deposit
Prompt for deposit amount.
Call the deposit() method to add the amount to the balance.
Case 2: Withdraw
Prompt for withdrawal amount.
Call the withdraw() method.
For SavingsAccount, deduct directly.
For CurrentAccount, deduct and check withdrawal limit.
Case 3: Show Balance
Display the current balance of the account.
Case 4: Compute Interest
Call the interest() method.
For SavingsAccount, compute and update the balance with interest.
For CurrentAccount, display a message that interest is not applicable.
Default: Exit
Exit the program.

3. End the Program
Close the Scanner object to release resources.
Terminate the program.

Example Execution Flow
User selects Savings Account.

User deposits 1000:

yaml

Copy code

```
Enter deposit amount: 1000
```

User computes interest:

csharp

Copy code

```
The interest is 50.0
```

User checks balance:

csharp

Copy code

```
The balance is 1050.0
```

User exits the program

program:

```
import java.util.Scanner;
import java.lang.Math;
class Bank{
```

```
int accountNo;
double balance;
Bank(int accountNo){
this.accountNo = accountNo;
this.balance = 0;
}
void deposit(double depAmount){
this.balance += depAmount;
}
void withdraw(double withAmount){
this.balance -= withAmount;
}
double interest(double rate, int time){
System.out.println("Interest is not applicable in current account");
return 0.0;
}
}
class SavingsAccount extends Bank{
SavingsAccount(int accountNo){
super(accountNo);
}
double interest(double rate, int time){
double interest = (balance * Math.pow((1 + (rate / 100)), time)) - balance;
balance += interest;
return interest;
}
}
class CurrentAccount extends Bank{
static double withdrawLimit = 1000;
CurrentAccount(int accountNo){
super(accountNo);
}
public void withdraw(double withAmount) {
super.balance -= withAmount;
if (balance < withdrawLimit) {
System.out.println("Withdraw Limit Reached - Deducting Service
Charge");
balance -= 100;
}
}
}
class Run {
public static void main(String[] args) {
System.out.println("Abhinav C 1BM23CS008");
double amount;
Scanner sc = new Scanner(System.in);}}}}
System.out.print("1. Open Savings Account\n2. Open Current
Account\n\nEnter Choice:");
```

```
int choice = sc.nextInt();
Bank acc;
if(choice == 1) {
acc = new SavingsAccount(101);
}
else {
acc = new CurrentAccount(201);
}
System.out.println("1. Deposit\n2. Withdraw\n3. Show Balance\n4. Compute
Interest\n5. Exit\n");
while(true){
System.out.print("Enter Choice: ");
choice = sc.nextInt();
switch(choice){
case 1: System.out.print("Enter deposit amount: ");
amount = sc.nextDouble();
acc.deposit(amount);
break;
case 2: System.out.print("Enter withdraw amount: ");
amount = sc.nextDouble();
acc.withdraw(amount);
break;
case 3: System.out.println("The balance is " + acc.balance);
break;
```

```
case 4: System.out.println("The interest is "+ acc.interest(5, 1));

break;

default:System.exit(0);
```

program 6:
Package
Algorithm:
Step 1: Define the Classes

Class: Student (Base Class):

Attributes:
name: Stores the name of the student.
marks: Array to store marks for 5 courses.
Methods:
Constructor: Initializes the name and an empty marks array.
getName(): Returns the student name.
setMarks(int[] marks): Updates the marks array.
getMarks(): Returns the marks array.

Class: Internal (Inherits from Student in package CIE)

Attributes:
internalMarks: Stores internal marks for 5 courses.
Methods:
Constructor: Accepts name and internalMarks, initializes them, and calls setMarks().

Class: External (Inherits from Student in package SEE)

Attributes:
externalMarks: Stores external marks for 5 courses.
Methods:
Constructor: Accepts name and externalMarks, initializes them, and calls setMarks().

Step 2: Main Program

Input the Number of Students

Prompt the user to input the total number of students (n).

Initialize Arrays

Create arrays of type Internal and External to hold n students' data.

Input Student Details

For each student (loop i from 0 to n-1):
Input the student's name.
Input internal marks for 5 courses and store in internalMarks array.
Input external marks for 5 courses and store in externalMarks array.
Create an Internal object for the student using the name and internalMarks.
Create an External object for the student using the name and externalMarks.

Display Final Marks

Print "Final Marks for all students".
For each student (loop i from 0 to n-1):
Retrieve internalMarks using getMarks() from the Internal object.
Retrieve externalMarks using getMarks() from the External object.
Print the student's name.
Print internalMarks and externalMarks.
Compute final marks for each course by adding corresponding elements of internalMarks and externalMarks.
Print the final marks.

End Program

Close the scanner to release resources.

code:

```
package SEE;
import CIE.Student;

public class External extends Student {
   private int[] externalMarks;


   public External(String name, int[] externalMarks) {
      super(name);
      this.externalMarks = externalMarks;
      this.setMarks(externalMarks);
   }


   public int[] getExternalMarks() {
      return externalMarks;
   }
```

```
    public void setExternalMarks(int[] externalMarks) {
        this.externalMarks = externalMarks;
        this.setMarks(externalMarks);
    }
}


package CIE;
import SEE.Student;

public class Internal extends Student {
    private int[] internalMarks;

    public Internal(String name, int[] internalMarks) {
        super(name);
        this.internalMarks = internalMarks;
        this.setMarks(internalMarks);
    }

    public int[] getInternalMarks() {
        return internalMarks;
    }

      public void setInternalMarks(int[] internalMarks) {
        this.internalMarks = internalMarks;
        this.setMarks(internalMarks);
    }
}


import CIE.Internal;
import SEE.External;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);


        System.out.print("Enter the number of students: ");
        int n = sc.nextInt();
        sc.nextLine();
```

```
Internal[] internalStudents = new Internal[n];
External[] externalStudents = new External[n];


for (int i = 0; i < n; i++) {
    System.out.print("Enter the name of student " + (i + 1) + ": ");
    String name = sc.nextLine();


    System.out.println("Enter internal marks (5 courses) for " + name + ": ");
    int[] internalMarks = new int[5];
    for (int j = 0; j < 5; j++) {
        internalMarks[j] = sc.nextInt();
    }
    sc.nextLine();


    System.out.println("Enter external marks (5 courses) for " + name + ": ");
    int[] externalMarks = new int[5];
    for (int j = 0; j < 5; j++) {
        externalMarks[j] = sc.nextInt();
    }
    sc.nextLine();


    internalStudents[i] = new Internal(name, internalMarks);
    externalStudents[i] = new External(name, externalMarks);
}


System.out.println("\nFinal Marks for all students:");
for (int i = 0; i < n; i++) {
    int[] internalMarks = internalStudents[i].getMarks();
    int[] externalMarks = externalStudents[i].getMarks();


    System.out.println("\nStudent: " + internalStudents[i].getName());


    System.out.print("Internal Marks: ");
    for (int mark : internalMarks) {
        System.out.print(mark + " ");
    }
```

```
            System.out.print("\nExternal Marks: ");
            for (int mark : externalMarks) {
               System.out.print(mark + " ");
            }


            System.out.print("\nFinal Marks: ");
            for (int j = 0; j < 5; j++) {
               int finalMark = internalMarks[j] + externalMarks[j];
               System.out.print(finalMark + " ");
            }
            System.out.println();
         }
      }
   }
```

Progam 7:
Exception

Algorithm:
Algorithm Steps
Step 1: Define the Custom Exceptions

NegativeAgeError Class:

Attributes:
a: Stores the invalid negative age.
Constructor:
Accepts an age value and initializes it.
toString() Method:
Returns a string representation indicating the age is negative.

InvalidAgeError Class:

Attributes:
a: Represents the son's age.
b: Represents the father's age.
Constructor:
Accepts the son's age and father's age and initializes them.
toString() Method:
Returns a string representation indicating the son's age is not less than the father's age.

Step 2: Define the Father Class
Attributes:
name: Stores the father's name.
age: Stores the father's age.
Constructor:
Accepts name and age.
If age < 0, throws NegativeAgeError.
Otherwise, initializes the name and age.

Step 3: Define the Son Class (Inherits from Father)
Attributes:
sonName: Stores the son's name.
sonAge: Stores the son's age.
Constructor:
Accepts sonName, sonAge, fatherName, and fatherAge.
Calls the Father constructor to initialize the father's details.

If sonAge < 0, throws NegativeAgeError.
If sonAge >= fatherAge, throws InvalidAgeError.
Otherwise, initializes the sonName and sonAge.

Step 4: Implement the Main Program
Print the program header (name and USN).
Create a Scanner object to accept user input.
Prompt the user for the son's name and age.
Prompt the user for the father's name and age.
Create a Son object using the entered details.
Handles any exceptions raised during object creation:
Prints the exception details using the custom toString() method of the exceptions.
Print "End of program."

code:

```
import java.util.Scanner;

class NegativeAgeError extends Exception {
   int a;

   public NegativeAgeError(int a) {
      this.a = a;
   }

   @Override
   public String toString() {
      return "Negative Age: " + a;
   }
}


class InvalidAgeError extends Exception {
   int a, b;

   public InvalidAgeError(int a, int b) {
      this.a = a;
      this.b = b;
   }

   @Override
   public String toString() {
      return "Invalid Age: " + a + " is less than " + b;
   }
}


class Father {
    String name;
    int age;

    Father(String name, int age) {
        try {
            if (age < 0) {
                throw new NegativeAgeError(age);
            }
            this.name = name;
            this.age = age;
        } catch (NegativeAgeError e) {
            System.out.println(e);
        }
    }
}

class Son extends Father {
    String sonName;
    int sonAge;

    Son(String sonName, int sonAge, String fatherName, int fatherAge) {
        super(fatherName, fatherAge);
        this.sonName = sonName;
        try {
            if (sonAge < 0) {
                throw new NegativeAgeError(sonAge);
            }
            if (sonAge >= fatherAge) {
                throw new InvalidAgeError(sonAge, fatherAge);
            }
            this.sonAge = sonAge;
        } catch (NegativeAgeError e) {
            System.out.println(e);
        } catch (InvalidAgeError e) {
            System.out.println(e);
        }
    }
}
```

```
class Exceptions {
   public static void main(String[] args) {
      System.out.println("Abhinav C\nUSN : 1BM23CS008");

      Scanner sc = new Scanner(System.in);

      System.out.println("Enter son name");
      String name = sc.next();

      System.out.println("Enter son age");
      int age = sc.nextInt();

      System.out.println("Enter father name");
      String fatherName = sc.next();

      System.out.println("Enter father's age");
      int fatherAge = sc.nextInt();

      Son a1 = new Son(name, age, fatherName, fatherAge);

      System.out.println("End of program");
   }
}
```

program 8:
Threads

Algorithm
Step 1:
Define the BMS Thread
Create a new class BMS that extends Thread class.
Override the run() method:
This method will contain the code to be executed by the thread.
In an infinite loop:
Print the message: "BMS College of Engineering".
Sleep for 10 seconds (10000 milliseconds) using Thread.sleep(10000).
If interrupted, catch the InterruptedException and print the exception message.
Step 2:
Define the CSE Thread
Create a new class CSE that also extends Thread.
Override the run() method:
This method will also execute an infinite loop.
Print the message: "CSE".
Sleep for 2 seconds (2000 milliseconds) using Thread.sleep(2000).
If interrupted, catch the InterruptedException and print the exception message.
Step 3:
Implement the Main Class (BMSCE)
Print the user's name and USN at the beginning of the program for identification.
Create instances of BMS and CSE threads:
Instantiate BMS bmsThread = new BMS(); and CSE cseThread = new CSE();.
Start both threads:
Call bmsThread.start(); to start the BMS thread.
Call cseThread.start(); to start the CSE thread.
Step 4:
Program Execution
The BMS thread will print "BMS College of Engineering" every 10 seconds.
The CSE thread will print "CSE" every 2 seconds.
Both threads will run concurrently and continuously print the messages with their respective sleep intervals.

code:
```
class BMS extends Thread {
    public void run() {
        while (true) {
            System.out.println("BMS College of Engineering");
            try {
                Thread.sleep(10000);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

class CSE extends Thread {
    public void run() {
        while (true) {
            System.out.println("CSE");
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

public class BMSCE {
    public static void main(String[] args) {

        System.out.println("Abhinav C\nUSN : 1BM23CS008");


        BMS bmsThread = new BMS();
        bmsThread.start();

        CSE cseThread = new CSE();
        cseThread.start();
    }
}
```

program 9:
inter process communication

algorithm:

Step 1: Define the Shared Resource Class Q

Create a class Q to act as the shared resource between producer and consumer threads.

Attributes:

int n: Stores the shared data.
boolean valueSet: Flag indicating whether the data (n) is available for consumption.

Define the get() method for the consumer:

Mark it as synchronized to ensure thread-safe access.
If valueSet is false (no data to consume):
Print a message indicating the consumer is waiting.
Call wait() to release the lock and wait for the producer to notify.
Print the value consumed (n) and set valueSet to false.
Notify the producer that it can produce more data using notify().
Return the consumed value.

Define the put() method for the producer:

Mark it as synchronized to ensure thread-safe access.
If valueSet is true (data is already available):
Print a message indicating the producer is waiting.
Call wait() to release the lock and wait for the consumer to notify.
Store the new value in n and set valueSet to true.
Print the value produced (n) and notify the consumer using notify().

Step 2: Define the Producer Class
Create a class Producer that implements Runnable:
It generates data to be shared with the consumer.
Constructor:
Accepts an instance of Q (shared resource) as a parameter.
Creates and starts a new thread for the producer: new Thread(this, "Producer").start();.

Override the run() method:
Use a loop (e.g., while(i < 15)) to produce data.
Call the put() method of Q to store the data and pass it to the consumer.

Step 3: Define the Consumer Class
Create a class Consumer similar to Producer but for consuming data:
Implements Runnable.
Constructor accepts the shared resource (Q) and starts a new thread.
Override the run() method:
Use a loop to call get() from the shared resource to consume data.

Step 4: Implement the Main Class
Create an instance of Q to act as the shared resource.
Create instances of Producer and Consumer, passing the shared resource (Q) to both:
Producer p = new Producer(q);
Consumer c = new Consumer(q);
Start the threads:
These threads will interact through the shared resource using wait() and notify() mechanisms.

code:

```
class Q {
int n;
boolean valueSet = false;
synchronized int get() {
while(!valueSet)
try {
System.out.println("\nConsumer waiting\n");
wait();
} catch(InterruptedException e) {
System.out.println("InterruptedException caught");
}
System.out.println("Got: " + n);
valueSet = false;
System.out.println("\nIntimate Producer\n");
notify();
return n;
}
synchronized void put(int n) {
while(valueSet)
try {
System.out.println("\nProducer waiting\n");
wait();
} catch(InterruptedException e) {
System.out.println("InterruptedException caught");
}
this.n = n;
valueSet = true;
System.out.println("Put: " + n);
System.out.println("\nIntimate Consumer\n");
notify();
}
}
class Producer implements Runnable {
Q q;
Producer(Q q) {
this.q = q;
new Thread(this, "Producer").start();
}
public void run() {
int i = 0;
while(i<15) {
q.put(i++);
}
}
}
```

program 10:

deadlock

Algorithm:

Step 1: Define Class A
Attributes and Methods:
foo(B b): A synchronized method in A that:
Prints a message indicating the thread has entered the method.
Sleeps for 1 second to simulate some work.
Attempts to call B.last() on the passed object b.
last(): A simple method that prints "Inside A.last".
Step 2: Define Class B
Attributes and Methods:
bar(A a): A synchronized method in B that:
Prints a message indicating the thread has entered the method.
Sleeps for 1 second to simulate some work.
Attempts to call A.last() on the passed object a.
last(): A simple method that prints "Inside B.last".
Step 3: Define Class Deadlock

Attributes:

A a: An instance of class A.
B b: An instance of class B.

Constructor:

Sets the current thread's name to "MainThread".
Creates and starts a new thread (RacingThread) that runs the run() method.
Calls a.foo(b) in the main thread, causing:
MainThread to lock object A.
MainThread to attempt calling B.last().

run() Method:

In the new thread (RacingThread), calls b.bar(a), causing:
RacingThread to lock object B.
RacingThread to attempt calling A.last().

Step 4: Main Method
Creates a new instance of the Deadlock class:
Starts the deadlock scenario with both threads executing simultaneously.

Execution Flow

Main Thread:

Calls a.foo(b):
Locks A.
Attempts to call B.last() but waits because RacingThread has locked B.

Racing Thread:

Calls b.bar(a):
Locks B.
Attempts to call A.last() but waits because MainThread has locked A.

Deadlock:

Both threads are stuck indefinitely because each is waiting for the other to release its lock.

Deadlock Scenario
The program results in a deadlock because:
MainThread locks A and waits for B (locked by RacingThread).
RacingThread locks B and waits for A (locked by MainThread).

<u>Code:</u>

```
class A {

  synchronized void foo(B b) {
    String name = Thread.currentThread().getName();
    System.out.println(name + " entered A.foo");

    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      System.out.println("A Interrupted");
    }
    System.out.println(name + " trying to call B.last()");
    b.last();
  }

  void last() {
    System.out.println("Inside A.last");
  }
}

class B {

  synchronized void bar(A a) {
    String name = Thread.currentThread().getName();
    System.out.println(name + " entered B.bar");

    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      System.out.println("B Interrupted");
    }

    System.out.println(name + " trying to call A.last()");
    a.last();
  }

  void last() {
    System.out.println("Inside B.last");
  }
}

class Deadlock implements Runnable {

  A a = new A();
  B b = new B();

  Deadlock() {
    Thread.currentThread().setName("MainThread");
    Thread t = new Thread(this, "RacingThread");
    t.start();

    a.foo(b);
    System.out.println("Back in main thread");
  }
```

```
   public void run() {
      b.bar(a);
      System.out.println("Back in other thread");
   }

   public static void main(String args[]) {
      new Deadlock();
   }
}
```